Towards an Ontological Concepts for Domain-Driven Software Design

Dr. Nedhal Al Saiyd
Applied Science University, Jordan
Nedhal_alsaiyd@asu.edu.jo

Dr. Intisar Al Said
Al-Isra University, Jordan
Intisar@ipu.edu.jo

Dr. Afaf Al Neaimi
Al-Zaytouna University, Jordan
drafafn@alzaytoonah.edu.jo

Abstract

Software design process has been followed and widely used to describe logical structure of software using different types of design model, to ensure consistency among the multiple views of a design. This paper describes an effort to identify the major concepts in software design that are required to understand system functionality from various perspectives. It also presents systematic framework ontology for software design lifecycle context, ontology development, and ontology representation. The aim is to resolve misunderstanding or misinterpretation especially with ambiguous terms for designers and users who have different backgrounds and knowledge of the software for functional-oriented, object-oriented, component-oriented, agent-oriented designs, and represent the reasons that led the designers to choose one design among the other alternatives.

1. Introduction

Architectural design is recognized as a critical element in the successful development of software systems. It influences the construction, deployment and evolution of the system as a whole. Designing domain-specific software artifact typically involves understanding the problem being addressed and user requirements, understanding analysis models, identifying possible design alternatives, analyzing them, and deciding which design will be used to construct the final artifact specifications [1].

There are different aspects of software design process such as data design, architectural design, interface design, and detailed design. Determining design activities require domain participant (i.e. designers and users) involvement in selecting good design plan, making design decisions, implementing design roadmap, selecting the best design out of other alternative solutions, and specifying design artifacts [2,3].

There is an increasing interest of knowledge in the application of architectural design concepts to achieve the benefits of reducing costs and improve quality, such as usability, flexibility, reliability, interoperability and other software qualities.

But the variety of design areas is huge, complicated and largely ill-defined. There is no guiding theory, no overall conceptual viewpoint and no uniform formalism. The concept of architecture have not been consistently defined and applied with lifecycle of software-intensive systems [2,3,4,5,6,7,8]. Despite significant industrial and research activity in this area, there is no single accepted framework codifying architectural thinking [2]. The final artifact represent part of the knowledge employed by designers during the design process, but do not represent the reasons that led the designers to choose that specific design model, and why the other alternatives are discarded. In other words, they do not capture the Design Rationale (DR). Design rationales include not only the reasons behind a design decision but also the justification for it and the argumentation that led to the decision. In most cases the DR is not adequately documented.

The fundamental motivation of our work is to resolve some of these difficulties and to come up with a systematic framework ontology that serves as a knowledge base for software design process. It incorporates the most common design methodologies utilized in object-oriented, function-oriented, component-based and multi-agent-oriented design processes, in addition to the DR activities. Representing software design process knowledge in the form of ontology is helping to clear up ambiguities in the terms used in the situation of software design process. The domain-specific ontological model addresses two orthogonal concerns; the first is, the general concepts and categories originating in the design knowledge, and the second is the specific behavior originating in DR and user requirements.

The remainder of this paper is organized as follows. The related works are presented in section 2. The suggested software design ontology model is introduced in section 3. The suggested ontology development is presented in section 4, then the ontology representation is presented in section 5. Section 5 presents ontology validation. Section 7 presents the conclusions and suggestions for the future work.

2. Related Works

There were some research attempts that concentrate on one aspect or issue of software design knowledge. Eden and Turner proposed hypothetically the ontology of software design. They suggested a combination of the intension and the locality criteria that divides the design statements into a hierarchy of three abstraction classes; Strategic, Tactical, and Implementation statements. The vocabulary they use in defining the criteria are described in mathematical logic [7]. Medeiros and et al [8] define rules that enable performing computable operations to support the use of design rational DR in the design process of new artifacts. Wongthongtham's and et al [9] aimed to present an ontology model of the software design structural and behavioral views (such as: entity-relation diagram, activity diagram, and state-chart diagram, etc.) to represent design knowledge. In [10] a first stage is described towards constructing of conceptual ontology for generic software architecture knowledge. They realize that semantic relationships between design terms are more difficult to construct and requires more work, and there are no documents that talk about scenarios, tactics, views and view types.

3. The Proposed Conceptual Framework for Software Design

The objective is to define an ontology describing the knowledge relevant to the practice of software design. The conceptualization step was based on a study a variety of design areas, activities and aspects, study of the literature [1,2,3,4,5,6,7,8], and the experience of the authors. Firstly, We study the related knowledge through three different and related areas as shown in figure 1. The first area is related to the knowledge of design participants (software designers); who make design decisions, select the best design to be implemented, and specify design models. The second area is related to design lifecycle context that can be classified further into the analysis models, DR and the aspects of software design process. The analysis model can be classified in turn into functional modeling, behavioral modeling and data modeling. They are considered as the input to the software design process. The third category is related to the design specification (i.e., structural and dynamic views). The three areas enable us to extract the useful knowledge through the identification of concepts, attributes, the relationships among concepts and the rules that govern these relationships. Secondly, we structure hierarchically the knowledge of software design into *design process* activities, which use *design techniques*, that deals with one or more of the *design strategies and methods*. The designer can find alternative designs, and *analyzes* them to choose the best one; which then specifies it using static and dynamic views. Guidelines in [11, 12] help us to formulate the software design hierarchically. Figure (2) shows the five main categories of software design, and the associated terms for each category are given beneath.

The conceptualization is the longest step and requires the definition of the scope of the software design ontology, definition of its concepts, description of each one (through a glossary, specification of attributes, domain values, and constraints). It represents the knowledge modeling itself. We identified some scenarios and questions that the ontology must answer. We started the ontology construction by looking for motivating scenarios and questions that help us in extracting the useful knowledge. Some of these scenarios are: deciding who is the best designer assigned to a design activity, based on designer skills and experience of the technology and the system considered; understanding requirement specification and the problem domain that the designer will act on (which are the analysis models and documents); defining the software design lifecycle context (i.e., design activities to be followed) in a specific software design style, and also the resources necessary to perform these activities.

These and other situations encourage us to organize the knowledge around four different aspects; which are proposed in a conceptual framework, as shown in figure(3).

The designer usually begins with a general question that establishes the problem to be solved. This general question can generate new questions that represent new design to solve sub-problems related to the main problem. For each question introduced, the designers can suggest ideas, formulating possible solutions to the problem expressed in the question. Ontologies are good candidate to represent software design life cycle activities and DR in a formally precise and computable way.

4. The Suggested Ontology Development

Ontology development is necessarily an iterative process and there is no one correct way to model a domain; there are always alternative models. The best solution almost always depends on the application and the anticipated extensions [11]. Then determining which software solution one would work better for the projected task, be more extensible, and more maintainable. Our experiences of ontology development have revealed and concluded that the processes should include the set of the following tasks:

Step 1: Identify the boundaries of software design. To identify the boundary of the domain, we have to create a simple lexicons or a controlled vocabulary which includes all the terms in this domain; by extracting 317 design concepts from the software engineering literatures [1,2,3]. Controlled vocabulary helps in eliminating meaningless terms, i.e. terms which are too broad or too narrow.

Step 2: Build the taxonomy: From the controlled vocabulary, we categorically organize a dictionary to build the taxonomy; classification that arranges the terms into super-class and sub-class hierarchy. Then define many relationships like *Related term, Uses, Consists of, has-a, is-a, described by, identifies, uses, work on, described by, ... etc*

Step 3: Identify attributes of classes and allowed values. Distinguishing properties are identified to define new concepts. These concepts have relationships with other concepts, to classify and clarify the classes. The binary relations are being used among classes, objects and data values that satisfy certain constraints. We organize classes, in a class hierarchy and create relationships among classes, in a similar way to that in [13, 14]. Then we applied UML class diagram successfully for some of the design methods as: functional-oriented, object-oriented, component-based and agent-based. Figure(4) shows a UML class diagram for a part of ontological concepts of software design methods and activities. It shows that software design is composite into software design process activities that has a sequence of design activities: data design, architectural design, interface design and unit design. The architectural design is decomposing the system into its components and identifying the relationships among them. The design methods can be either functional-oriented, object-oriented, component-based or agent-based. The design methods is described by static and dynamic views. The design method uses the design techniques that has the instances; decomposition, cohesion, abstraction, coupling and extensibility.

Step 4: Define axioms and rules for constraint checking. To define the constraints over the concepts and relations, we defined axioms in first order logic (FOL).

To express the constraints over the relations (e.g. correlation or realization), we defined a set of axioms like $\forall$ (a,b) correlation(a,b) $\wedge$ ReqSpec(a) $\rightarrow$ ReqSpec(b)). It specifies that: if $a1$ is a requirement specification and $a1$ correlated to $a2$, then $a2$ must also be a requirement specification (i.e. the correlation relation stands between artifacts of the same type), and $\forall$ (a,b) (realization(a,b) $\wedge$ ReqSpec(a) $\rightarrow \neg$ ReqSpec(b)). Similarly, the second axiom specifies that realization may only stand between two artifacts of different kind.

5. Ontology Representation

As it has explained in step 4, the *ontology representation or formalization* was done using first order logic; which uses a well-formed formula (wff). It is a sentence containing no "free" variables. i.e., all variables are bounded by universal or existential quantifiers.

The designer must *understand* the concepts of the application domain and the tasks performed in it. To express those relations, we defined axioms like: ($\forall$ d) (designer(d)$\rightarrow \exists$ (t) (CompScTechnoloogy(t) $\wedge$ knows(d,t)), and ($\forall$ d) (designer(d) $\rightarrow$ ($\exists$ a) (DesignActivity(a) $\wedge$ knows(m,a)); to express that any designer knows at least one technology and one activity).

The axioms are also used to specify the sequence or ordered of software design activities: $\forall$ (a, b) (pre-activity (a,b)$\rightarrow \neg$ pre-activity (b,a)) expressing the anti-symmetry, and $\forall$(a,b,c) (pre-activity(a,b) $\wedge$ pre-activity(b,c) $\rightarrow$ pre-activity(a,c), expressing the transitivity on the design activities.

6. Ontology Validation

With the ontology defined, we started its validation in two ways: validation of the quality of the ontology itself (how clear it is, how complete, concise, etc.), and validation of the usefulness of the concepts for maintenance (which was the ontology's purpose). To validate the quality of the ontology we considered (a) consistency, referring to the absence (or not) of contradictory information in the ontology; (b) completeness, referring to how well the ontology covers the real world (software design for us); (c) clarity, referring to how effectively the intended meaning is communicated. We present our ontological model to three high-skilled software engineers to evaluate it. The evaluation is good. The usefulness of concepts of the Design Process ontology appears when it can give answers for some of the important questions, like: what are the design types? What are the design methods? What are their possible sources? What are the activities performed during design? What does one need to perform them? Who perform them? What do they produce?. Also it can answer What kind of procedures (methods, techniques, and styles) does the designer know? What programming techniques and/or modeling languages does the designer know?

But we observe that some concepts that are related with domain applications, design tools, modeling languages, design guidelines; are not well-identified or not exist in the ontology for lack of enough examples and there were fewer sessions in our first experiment.

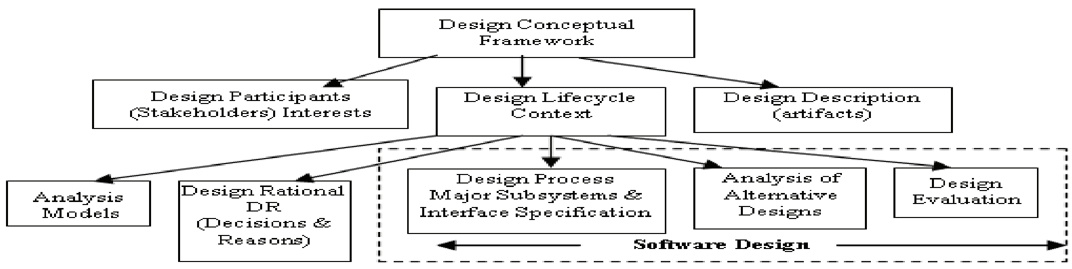

Design Conceptual Framework
Design Participants (Stakeholders) Interests
Design Lifecycle Context
Design Description (artifacts)
Analysis Models
Design Rational DR (Decisions & Reasons)
Design Process Major Subsystems & Interface Specification
Analysis of Alternative Designs
Design Evaluation
Software Design

Figure (1) The hierarchical of the three different areas related to software design

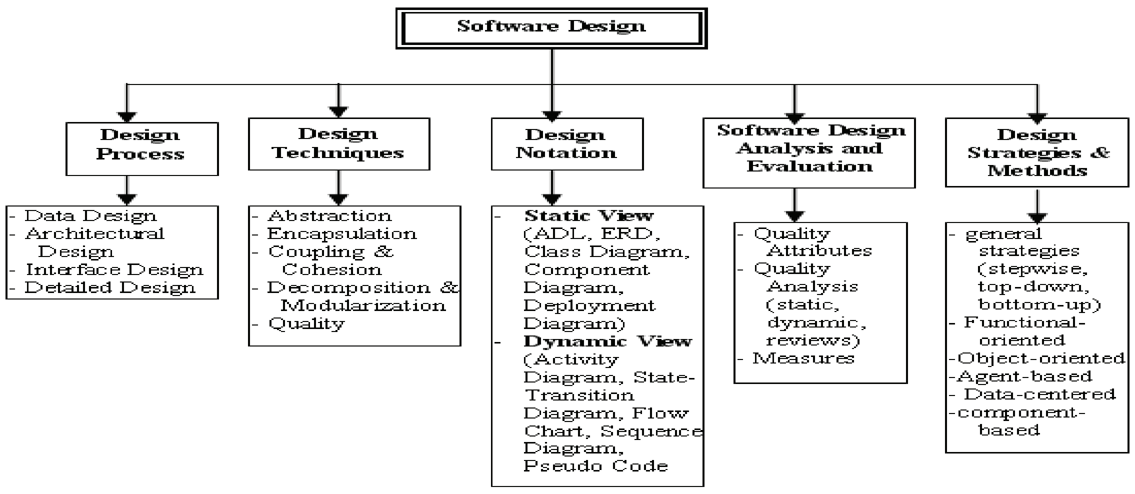
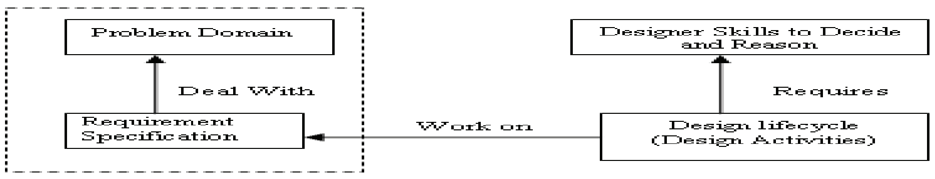

Software Design
Design Process
- Data Design
- Architectural Design
- Interface Design
- Detailed Design
Design Techniques
- Abstraction
- Encapsulation
- Coupling & Cohesion
- Decomposition & Modularization
- Quality
Design Notation
- Static View (ADL, ERD, Class Diagram, Component Diagram, Deployment Diagram)
- Dynamic View (Activity Diagram, State-Transition Diagram, Flow Chart, Sequence Diagram, Pseudo Code
Software Design Analysis and Evaluation
- Quality Attributes
- Quality Analysis (static, dynamic, reviews)
- Measures
Design Strategies & Methods
- general strategies (stepwise, top-down, bottom-up)
- Functional-oriented
-Object-oriented
-Agent-based
- Data-centered
-component-based

Figure (2) The five main categories of software design, and their associated terms


Problem Domain
Deal With
Requirement Specification
Work on
Designer Skills to Decide and Reason
Requires
Design lifecycle (Design Activities)

Figure(3) Ontology overview

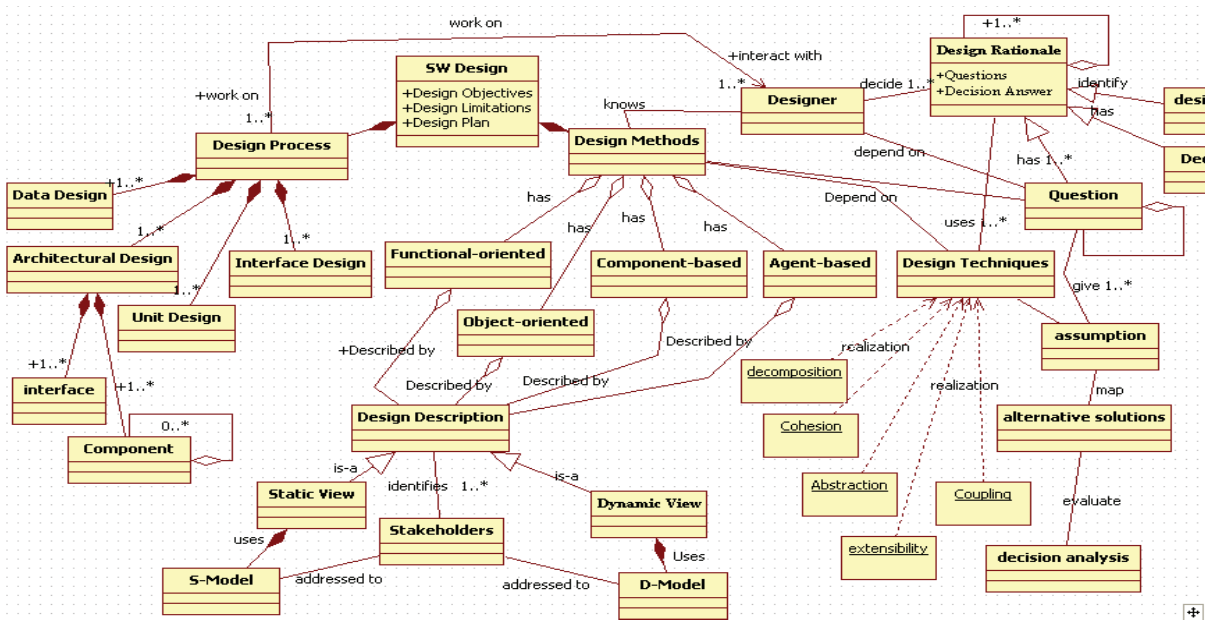


Figure (4) A portion of ontological concepts of software design methods and activities

7. Conclusions and Further Work

This research contributes to software design in several ways. **First**, our study categorizes the software design concepts into a framework. Taxonomy or ontology of the software design enables the designer in understanding the best practices and the relationships between them and also provide a means to apply them to the software systems to be developed. **Second**, an ontology development steps have been suggested to build a knowledge base of classes, objects and attributes. This knowledge base will help the designer to analyze the designed software system. **Third**, to better manage a large number of design models, we propose that an ontological approach for storing and searching design concepts is necessary. **Fourth**, the first order predicate is used to represent software design axioms.

This approach gives us the ability to create views on domain-driven demand from a structured repository, and the ontology itself can be easily extended, e.g. by adding new concepts. The approach provides a common vocabulary to enhance precision, usefulness and clarity that provides better design decision making, and its justification.

Future research will include completing the construction of a knowledge base of software design ontologies, and making more validations to increase the quality of ontology. Based on our approach, we will further examine ways to infer the relationships among design concepts and design actions by using semantic information**.**

Acknowledgments

The authors would like to thank the Applied Sciences University in Amman, Jordan for supporting this publication.

References

[1] IEEE Recommended Practice for Architectural Description of Software-Intensive Systems, IEEE STD 1471-2000, IEEE (2000).

[2] Pressman, R. S. & Ince Darre, Software Engineering a Practitioner's Approach: European Adaptation, McGraw Hill, 2006.

[3] Summerville, I. Software Engineering, Addison-Wesley, 7th Edition, 2006.

[4] SWEBOK: Guide to the Software Engineering Body of Knowledge, A project of the IEEE Computer Society Professional Practices Committee, Version 2004, available from:www.geocities.com/lbu_measure/swebok/swebok.htm

[5] Akerman A. , and Tyree J., Using Ontology to Support Development of Software Architectures**,** IBM Systems Journal; Oct 1, 2006

[6] Witmer, G., Dictionary of philosophy of mind–ontology, 2004. Available from: http://www.artsci.wustl.edu/~philos/MindDict/ontology.html

[7] Eden A. H., and Turner R. Towards an Ontology of Software Design: The Intension/Locality hypothesis, 3rd European Conf. Computing And Philosophy—ECAP (2-4 Jun. 2005), Sweden.

[8] Medeiros, A. P.; Schwabe, D.; Feijo, B. "Kuaba Ontology: Design Rationale Representation and Reuse in Model-Based Designs", Proceedings of the 24th International Conference on Conceptual Modeling (ER 2005), Klagenfurt, Austria, Lecture Notes in Computer Science, pp 241-255, Springer, October 2005.

[9] Wongthongtham, Pornpit and Chang, Elizabeth and Dillon, Tharam, Software Design Process Ontology Development, in Meersman, R. and Tari, Z. and Herrero, P. (ed), 2nd IFIP WG 2.12 & WG 12.4 International Workshop on Web Semantics (SWWS) in conjunction with OTM 2006, Oct 29 2006, pp. 1806-1813. Montpellier, France: Springer-Verlag.

[10] Babu T L., Ramaiah M S, Prabhakar T.V., Rambabu D, ArchVoc – Towards an Ontology for Software Architecture, 29th International Conference on Software Engineering Workshops(ICSEW'07)

[11] N. F. Noy, and D. L. McGuinness, "Ontology Development 101: A Guide to Creating Your First Ontology. Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880", March 2001. Available from: http://mia.ece.uic.edu/~papers/MediaBot/ontology101.pdf

[12] Siricharoen W V., Ontology Modeling and Object Modeling in Software Engineering, International Journal of Software Engineering and Its Applications Vol. 3, No. 1, January, 2009

[13] P. Kogut, S. Cranefield, L. Hart, M. Dutra, K. Baclawski, M. Kokar, and J. Smith, UML for Ontology Development, The Knowledge Engineering Review, Vol 17, Issue 1, March 2002, pp(61-64), Cambridge University Press.

[14] J. Iris Reinhartz-Berger1, Arnon Sturm2, and Yair Wand1 , Akoka et al. (Eds.): Domain Engineering – Using Domain Concepts to Guide Software Design, ER Workshops Springer-Verlag Berlin Heidelberg, LNCS 3770, pp. 461 – 463, 2005.